# SYSC 4001 - Assignment 3 Report Part 1

Student 1: Ajay Uppal
ID: 101308579
Student 2: Paul Felfli
ID: 101299663
Date: December 1 2025
Instructor: Professor Gabriel Wainer

**Introduction**

This simulation compares the three scheduling algorithms (EP, RR, and EP_RR) across CPU-bound, I/O-bound, and mixed workloads. Each test case demonstrates how different scheduling behaviors affect process wait time, turnaround time, responsiveness, and overall throughput. By looking at the state transitions by each algorithm, we can see their strengths and weaknesses and understand how workload characteristics influence scheduling performance.

## Simulation Results

CPU-Bound Workload (tc05_many_short_cpu_same_arrival)

This workload contains several short CPU bursts with no I/O operations, and all processes arrive at the same time. Since no process ever blocks, the scheduler that minimizes context switching performs the best.

EP ends up running each process almost right after it arrives and lets it finish in one go. Since all of the CPU bursts are basically the same length, EP ends up acting a lot like FCFS. It gives predictable turnaround times and doesn't waste time on extra scheduling overhead.

RR introduces unnecessary context switches. Every short CPU burst is sliced into time-quantum segments, which increases waiting time and slightly lowers throughput. Still, since all jobs are short, RR remains fair but less efficient.

EP_RR sits in between. It inherits RR's responsiveness but also respects priority ordering, reducing some of the unnecessary switching. Throughput is slightly better than RR but still below EP.

SYSC4001_A3_P1 / input_files / tc05_many_short_cpu_same_arrival.txt

felpau05 PCB

Code Blame Executable File · 5 lines (5 loc) · 90 Bytes

```
1, 5, 0, 20, 0, 0
2, 5, 0, 20, 0, 0
3, 5, 0, 20, 0, 0
4, 5, 0, 20, 0, 0
5, 5, 0, 20, 0, 0
```

SYSC4001_A3_P1 / output_files / tc05_many_short_cpu_same_arrival_EP.txt

felpau05 Fixes to all_process_terminated. Ran all Tests.

Code Blame Executable File · 24 lines (24 loc) · 1.21 KB

```
+------------------------------------------------+
|Time of Transition |PID | Old State | New State |
+------------------------------------------------+
|                 0 |  1 |NOT_ASSIGNED |     NEW |
|                 0 |  1 |       NEW |     READY |
|                 0 |  2 |NOT_ASSIGNED |     NEW |
|                 0 |  2 |       NEW |     READY |
|                 0 |  3 |NOT_ASSIGNED |     NEW |
|                 0 |  3 |       NEW |     READY |
|                 0 |  4 |NOT_ASSIGNED |     NEW |
|                 0 |  4 |       NEW |     READY |
|                 0 |  5 |NOT_ASSIGNED |     NEW |
|                 0 |  5 |       NEW |     READY |
|                 0 |  1 |     READY |   RUNNING |
|                19 |  1 |   RUNNING |TERMINATED |
|                20 |  2 |     READY |   RUNNING |
|                39 |  2 |   RUNNING |TERMINATED |
|                40 |  3 |     READY |   RUNNING |
|                59 |  3 |   RUNNING |TERMINATED |
|                60 |  4 |     READY |   RUNNING |
|                79 |  4 |   RUNNING |TERMINATED |
|                80 |  5 |     READY |   RUNNING |
|                99 |  5 |   RUNNING |TERMINATED |
+------------------------------------------------+
```

I/O-Bound Workload (tc13_long_io_duration)

tc13 contains long CPU bursts but extremely long I/O durations that repeatedly move processes into the WAITING state.

EP doesn’t perform well in this situation. Since each process spends a long time blocked on I/O, EP ends up rotating through the READY queue very slowly. When processes finish their I/O and return to READY, their priority order doesn’t always line up with the order in which they completed I/O. This creates extra waiting after every I/O operation and leads to longer overall delays.

RR handles this workload significantly better. When a process unblocks after a long I/O, RR quickly gives it CPU time due to its strict time-slice fairness. This reduces response time and avoids starvation, especially when multiple processes re-enter READY close together.

EP_RR gives the best performance. It keeps RR’s responsiveness but also uses priority to avoid inefficient rotations. The transitions show frequent READY → RUNNING changes immediately after I/O completion, indicating the CPU is rarely idle and the scheduler can adapt well to the blocking pattern.

SYSC4001_A3_P1 / input_files / tc13_long_io_duration.txt

felpau05 PCB

Code Blame Executable File · 3 lines (3 loc) · 69 Bytes

```
1, 10, 0, 500, 50, 100
2, 10, 0, 500, 50, 100
3, 10, 0, 500, 50, 100
```

SYSC4001_A3_P1 / output_files / tc13_long_io_duration_EP.txt

felpau05 Fixes to all_process_terminated. Ran all Tests.

Code Blame Executable File · 97 lines (97 loc) · 4.84 KB

```
+------------------------------------------------+
|Time of Transition |PID | Old State | New State |
+------------------------------------------------+
|                 0 |  1 |NOT_ASSIGNED |       NEW |
|                 0 |  1 |         NEW |     READY |
|                 0 |  2 |NOT_ASSIGNED |       NEW |
|                 0 |  2 |         NEW |     READY |
|                 0 |  3 |NOT_ASSIGNED |       NEW |
|                 0 |  3 |         NEW |     READY |
|                 0 |  1 |       READY |   RUNNING |
|                49 |  1 |     RUNNING |   WAITING |
|                50 |  2 |       READY |   RUNNING |
|                99 |  2 |     RUNNING |   WAITING |
|               100 |  3 |       READY |   RUNNING |
|               149 |  1 |     WAITING |     READY |
|               149 |  3 |     RUNNING |   WAITING |
|               150 |  1 |       READY |   RUNNING |
|               199 |  2 |     WAITING |     READY |
|               199 |  1 |     RUNNING |   WAITING |
|               200 |  2 |       READY |   RUNNING |
|               249 |  3 |     WAITING |     READY |
|               249 |  2 |     RUNNING |   WAITING |
|               250 |  3 |       READY |   RUNNING |
|               299 |  1 |     WAITING |     READY |
|               299 |  3 |     RUNNING |   WAITING |
|               300 |  1 |       READY |   RUNNING |
|               349 |  2 |     WAITING |     READY |
|               349 |  1 |     RUNNING |   WAITING |
|               350 |  2 |       READY |   RUNNING |
|               399 |  3 |     WAITING |     READY |
|               399 |  2 |     RUNNING |   WAITING |
```

Mixed CPU/I-O Workload (tc08_mixed_cpu_io_staggered)

tc08 includes processes that differ in their CPU burst length, I/O usage, and staggered arrival patterns. This workload tests fairness, responsiveness, and preemption effectiveness.

EP performs reasonably well, but its strict priority order sometimes delays lower-priority tasks longer than necessary. When a higher-priority process returns from I/O, it can repeatedly preempt others, increasing wait time for some jobs.

RR works well but has issues from overhead. Because many transitions happen between RUNNING → WAITING → READY, RR's constant preemption increases turnaround time for longer CPU bursts. However the response time remains high because every process gets early CPU access.

EP_RR provides the most effective performance for this mixed workload. It reduces the high context-switching overhead seen in RR while also avoiding the extended delays that can occur under EP's strict priority structure. The scheduling transitions show that processes returning from I/O are able to resume execution promptly, without causing unnecessary disruption to the overall schedule. This balanced behaviour leads to more consistent waiting times and improved overall throughput.

felpau05 PCB

Code Blame Executable File · 4 lines (4 loc) · 89 Bytes

```
1, 10, 0, 200, 0, 0
2, 10, 50, 200, 50, 10
3, 10, 100, 200, 80, 20
4, 10, 150, 200, 0, 0
```

SYSC4001_A3_P1 / output_files / tc08_mixed_cpu_io_staggered_EP.txt

felpau05 Fixes to all_process_terminated. Ran all Tests.

Code Blame Executable File · 35 lines (35 loc) · 1.75 KB

```
+------------------------------------------------+
|Time of Transition |PID | Old State | New State |
+------------------------------------------------+
|                 0 |  1 |NOT_ASSIGNED |       NEW |
|                 0 |  1 |        NEW |     READY |
|                 0 |  1 |      READY |   RUNNING |
|                50 |  2 |NOT_ASSIGNED |       NEW |
|                50 |  2 |        NEW |     READY |
|               100 |  3 |NOT_ASSIGNED |       NEW |
|               100 |  3 |        NEW |     READY |
|               150 |  4 |NOT_ASSIGNED |       NEW |
|               150 |  4 |        NEW |     READY |
|               199 |  1 |    RUNNING |TERMINATED |
|               200 |  2 |      READY |   RUNNING |
|               249 |  2 |    RUNNING |   WAITING |
|               250 |  3 |      READY |   RUNNING |
|               259 |  2 |    WAITING |     READY |
|               329 |  3 |    RUNNING |   WAITING |
|               330 |  2 |      READY |   RUNNING |
|               349 |  3 |    WAITING |     READY |
|               379 |  2 |    RUNNING |   WAITING |
|               380 |  3 |      READY |   RUNNING |
|               389 |  2 |    WAITING |     READY |
|               459 |  3 |    RUNNING |   WAITING |
|               460 |  2 |      READY |   RUNNING |
|               479 |  3 |    WAITING |     READY |
|               509 |  2 |    RUNNING |   WAITING |
|               510 |  3 |      READY |   RUNNING |
|               519 |  2 |    WAITING |     READY |
|               549 |  3 |    RUNNING |TERMINATED |
|               550 |  2 |      READY |   RUNNING |
|               599 |  2 |    RUNNING |TERMINATED |
|               600 |  4 |      READY |   RUNNING |
|               799 |  4 |    RUNNING |TERMINATED |
+------------------------------------------------+
```

## Conclusion

The results show that each scheduling algorithm performs best under different workload conditions. EP is most effective for CPU-bound tasks due to its low overhead, while RR handles I/O-bound workloads better because of its fairness and quick response to unblocked processes. EP_RR gives the most balanced performance overall, maintaining reasonable waiting times and strong throughput across any mixed workloads. What we can conclude with these results is that there isn’t a specific scheduler that would be ideal for every single scenario but the performance depends on how the process is being executed.